PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 13/00

A1

(11) International Publication Number: WO 00/25223

(43) International Publication Date: 4 May 2000 (04.05.00)

(21) International Application Number: PCT/US99/25159

(22) International Filing Date: 27 October 1999 (27.10.99)

(30) Priority Data:
09/179,790 27 October 1998 (27.10.98) US

(71) Applicant: CUSTOMER POTENTIAL MANAGEMENT CORPORATION [US/US]; Suite #2, 2500 North Main Street, East Peoria, IL 61615 (US).

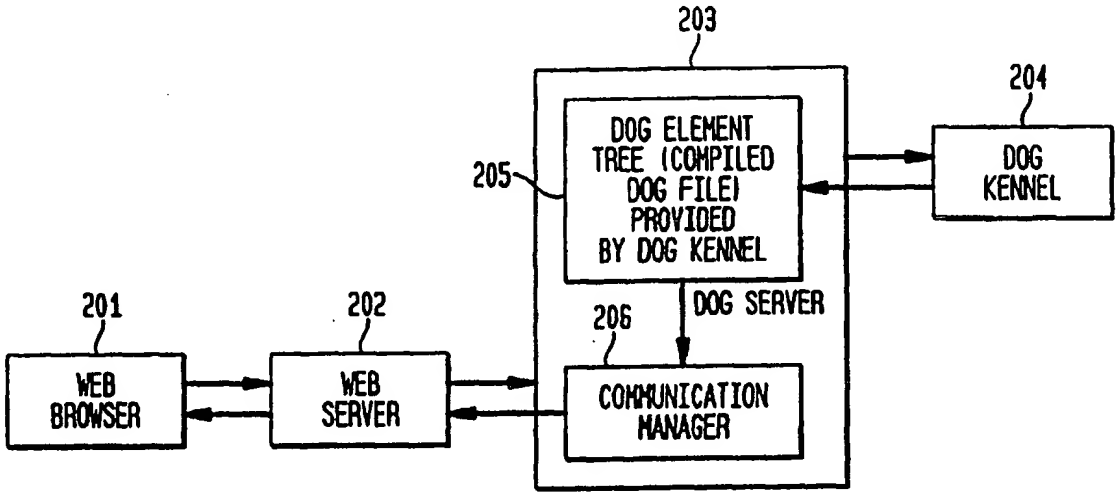
(72) Inventor: BOURRILLION, Kevin; Apartment #1110, 5908 North Cypress Drive, Peoria, IL 61615 (US).

(74) Agent: AHN, Harry, K.; Baker & McKenzie, 805 Third Avenue, New York, NY 10022 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: METHOD AND APPARATUS FOR GENERATING DYNAMIC WEB PAGE AND INTERFACING EXTERNAL SYSTEMS



(57) Abstract

A method for generating dynamic output from a source includes providing a web browser request to a web server (202), the web browser determines whether a dog file is being requested, and if the dog file is not being requested, the web server provides a response to the web browser (201). If the dog file is being requested, the web server provides a web server request to a dog server (203) based on the web browser request. The dog server provides a dog file extraction request to a dog kennel (204). The dog kennel returns a copy of a compiled dog file to the dog server (203). The dog server (203) executes the compiled dog file and provides the dynamic output to the web browser (201).

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

METHOD AND APPARATUS FOR GENERATING A DYNAMIC WEB PAGE AND INTERFACING EXTERNAL SYSTEMS

Reference To Paper Appendix

The present application includes a paper appendix attached hereto setting forth exemplary core dog tags of an exemplary embodiment of the present invention which is hereby incorporated by reference. A portion of the disclosure of the present application includes material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent disclosure, as it appears in the Patent & Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

Field Of The Invention

The present invention relates to the generation of dynamic output from data driven instructions, such as generating dynamic web pages from web pages embedded with predefined tags, and interfacing with other systems such as database systems, file systems and email systems from the dynamic web page.

Background Information

A way of communicating information on the World Wide Web (WWW) is through web pages. A web browser allows a user to request a web page from a web server, for example, by specifying a universal resource locator (URL). A URL is a pointer to a

resource, for example, a web page located on a network such as the WWW. Conventional web pages are created through the use of Hypertext Markup Language (HTML). HTML is an authoring software language for creating web pages. Conventional web pages, however, are generally stored as static files.

Although utilizing web pages that are stored as static files has generally been accepted, a need for providing information which is dynamically generated and the ability to alter and manipulate other systems in an efficient and costly manner is needed.

Summary Of The Invention

An object of the present invention is providing a method and apparatus for providing dynamic output from other systems.

It is another object of the present invention to provide a method and apparatus for communicating with other systems for the manipulation and alteration of those systems.

Another object of the present invention provides a method for creating tags which provide dynamic output information from other systems.

An aspect of the present invention provides a method for generating dynamic output from a system. The method includes a web browser providing a web browser request, e.g., a hypertext transfer protocol (http) request, to a web server. The method

also includes the web server determining whether a dog file, e.g., a file that includes components that can be compiled into specific implementations of the requirement that an execute method that operates on a context and a symbol table while returning a status exists, is being requested by the web browser request. If the dog file is not being requested, the web server provides to the web browser a response to the web browser request. If the dog file is being requested, the web server provides a web server request to a dog server. In this case, the dog server provides a dog file extraction request to a dog kennel, the dog kennel provides a copy of a compiled dog file to the dog server, and the dog server executes the compiled dog file and provides the dynamic output as a result of the execution of the compiled dog file to the web browser.

Another aspect of the present invention provides an apparatus for generating dynamic output from a system. The apparatus includes a web browser, a web server, a dog server, and a dog kennel. The web browser provides a web browser request to the web server, the web server determines whether a dog file is being requested by the web browser request, and if the dog file is not being requested, the web server provides to the web browser a response to the web browser request. If, however, the dog file is being requested, the web server provides a web server

request to the dog server based on the web browser request. In this case, the dog server provides a dog file extraction request to the dog kennel, the dog kennel provides a copy of a compiled dog file to the dog server, and the dog server executes the copy of the compiled dog file and provides the dynamic output to the web browser.

In yet another aspect of the present invention, a method for creating a dog tag is provided which is associated with a class and a type. The method includes providing a unique tag name for the dog tag, and determining the type of the dog tag. The method also includes assigning the dog tag to a package and extending the class for the type of the dog tag. The method also includes implementing an execute method for the dog tag, importing a library, and providing a constructor. The method further includes creating a file implementing the steps of providing a unique tag name, and determining the type of the dog tag. The created file also implements the steps of assigning the dog tag to the package and extending the class of the type of the dog tag. The method further includes implementing the execute method, importing a library, and providing the constructor.

Brief Description Of The Drawings

FIG. 1 illustrates a functional block diagram of a conventional computer system.

FIG. 2 illustrates an exemplary embodiment of a functional block diagram of a system according to an embodiment of the present invention.

FIG. 3 illustrates an exemplary flowchart for a web browser providing a request to a web server according to an embodiment of the present invention.

FIG. 4 illustrates an exemplary flowchart for a web server processing a request from a web browser according to an embodiment of the present invention.

FIG. 5A illustrates an exemplary flowchart for a dog server processing a request from the web server according to an embodiment of the present invention.

FIG. 5B illustrates an exemplary flowchart for a dog kennel processing a dog file extraction request from a dog server according to an embodiment of the present invention.

FIG. 6 illustrates an exemplary flowchart for a dog server processing a compiled dog file provided by a dog kennel according to an embodiment of the present invention.

FIG. 7 illustrates an exemplary flowchart for a web browser interpreting a response to its request according to an embodiment of the present invention.

FIG. 8 illustrates an exemplary flowchart for creating a custom dog tag according to an embodiment of the present invention.

Detailed Description

FIG. 1 illustrates a conventional computer system 101 in which the present invention operates. In an exemplary embodiment, the present invention may be implemented on SUN™ Workstations manufactured by SUN MICROSYSTEMS™, IBM™ Personal Computers manufactured by IBM Corporation and/or a MACINTOSH™ computers manufactured by APPLE™ Computer. In a preferred embodiment, the present invention is implemented on computer systems operating on a UNIX platform and Java virtual machine. Further, the computer systems may communicate with each other through a data communication network such as the Internet (not shown). It will be apparent to those of ordinary skill in the art that other computer system architectures may also be employed.

In general, such computer systems as illustrated by FIG. 1 includes a bus 102, for example for communicating information, a processor 103 such as a central processing unit coupled with the bus 102 for processing information, and a main memory 104 coupled with the bus 102 for storing information and instructions for the processor 103. A read-only memory 105 is coupled with the bus 102 for storing static information and instructions for the processor 103. A display device 106 coupled with the bus 102 displays information for a developer and an alphanumeric input

device 107 coupled with the bus 102 communicates information and command selections to the processor 103. A modem 110 is coupled with the bus 102 provides communication with, for example, other computer systems or databases and a mass storage medium 108, such as a magnetic disk and associated disk drive coupled with the bus 102 for storing information and instructions. A data storage medium 109 containing digital information is configured, for example to operate with a mass storage medium 108 to allow processor 103 access to the digital information on data storage medium 109 via bus 102. In addition, a CD-ROM drive (not shown) may also be used for the storage of high resolution images for display on the display device 106.

An embodiment of the present invention is implemented, for example, as a software module written in JAVA which may be executed on a computer system such as computer system 101 in a conventional manner. In an exemplary embodiment of the present invention, the database connectivity is written to the JAVA Database Connectivity Standard (JDBC) to ensure it will connect successfully to any JDBC-compliant database server. Using well known techniques, the application software of the preferred embodiment is stored on data storage medium 109 and subsequently loaded into and executed within the computer system 101. Once

initiated, the software of the preferred embodiment operates, for example, in the manner described below.

In an exemplary embodiment of the present invention, a dog element template is a description of the requirement of an execute method that operates on a context and a symbol table while returning a status exists. A dog element, for example, is a specific implementation of the dog element template, e.g., a dog element is a specific implementation of the requirement that an execute method that operates on a context and a symbol table while returning a status exists.

In an exemplary embodiment of the present invention, a dog tag is a tag that is identified and processed to create a dog element, e.g., a tag that is identified and processed to create a specific implementation of the requirement that an execute method that operates on a context and a symbol table while returning a status exists. In an exemplary embodiment of the present invention, dog tags may include a beginning tag, a dog tag body, and an end tag. The beginning tag may include a tag name and optional parameters. The dog tag body may include other dog tags. The syntax of the dog tag and a variable are different. In an exemplary embodiment of the present invention, the dog tag syntax is based on HTML with an additional identifying character located within the tag, for example, an exclamation point ("!")

located at the beginning of the tag name. The identifying character identifies the tag as a dog tag which should be executed by the parser, for example, the dog server 203.

In an exemplary embodiment of the present invention, there are several major types of dog tags which include a first type of dog tag, e.g., a tag that includes a beginning tag, and optional parameters that is compiled into a standalone tag. A second type of dog tag includes a beginning tag, a dog tag body, an end tag and optional parameters that is compiled into a container tag. A standalone tag performs execution based on the optional parameters defined by the tag. In contrast, a container tag performs execution based on the optional parameters and the dog element body of the container tag. The dog element body is the dog tag body of the second type of dog tag after it has been compiled. The dog element body, for example, of the container tag may include dog elements which may also get executed while the container tag is being executed.

In an exemplary embodiment of the present invention, the dog element body may include a plurality of dog elements. Each container tag includes, for example, a means for compiling a dog tag body to a dog element body. Further, the execution of a container tag executes all the tags embodied within it. In an exemplary embodiment of the present invention, there are several

classes of variables which include reserved variables (variables preset by the dog server and web server request) and user-defined variables (variables set and modified by the initial web browser request and execution of the dog elements). Further, there are several types of variables for each of these classes which include text variables, list variables and file variables. In an exemplary embodiment of the present invention, a dog file is a file that includes components that can be compiled into specific implementations of the requirement that an execute method that operates on a context and a symbol table while returning a status exists. A dog file may be a text file designated with a dog file extension such as ".dog". In an exemplary embodiment of the present invention, the dog file is a UNIX file. The dog file may be, for example, a static file that includes dog tags and standard hypertext markup language (HTML).

In an exemplary embodiment of the present invention, a dog server 203 is a server that requests and executes the plurality of dog elements, e.g., a server that requests and executes specific implementations of the requirement that an execute method operates on a context and a symbol table while returning a status exists. A dog kennel 204 performs compilation on the dog file, stores the tree of dog elements (compiled dog file), and

provides a copy of the compiled dog file to the dog server 203 to be executed on.

An implementation of a requirement can be utilized without having knowledge of the specific implementation. Thus, in an exemplary embodiment of the present invention, a server can utilize the execution method of the dog element to generate dynamic output through interactions with external systems without having to know the implementations of that method.

FIG. 2 illustrates a block diagram of an exemplary embodiment of the present invention. Each of the web browser 201, web server 202, dog server 203 including a communication manager 206, and dog kennel 204 may be provided on, for example, conventional computer systems 101 as shown in FIG. 1. As shown in FIG. 2, web browser 201 requests information from a web server 202 based upon an http request. When web browser 201 makes a request, it includes not only the desired web page, but also additional information necessary to process the request. For example, the additional information may include a user identification, password, and any other information collected by the web browser.

The web server 202 processes the request from web browser 201. Based upon the request, the web server 202 will provide either a file to the web browser 201 or a request for a dog file

to a dog server 203. In the case of a web server 202 providing a request to the dog server 203, the web server 202 may also provide a point of entry for receiving a response to the request of the web browser 201.

Referring to the block diagram of FIG. 2, upon receiving the request from the web server 202, the dog server 203 processes it. The dog server 203 provides the dog kennel 204 with a dog file extraction request. The dog kennel 204 serves as a central repository for storing the existing compiled dog files 205, compiling new dog files, and storing the newly compiled dog files therein. The dog kennel 204 processes the extraction request and provides the dog server 203 with a copy of the tree of dog elements (compiled dog file) 205. The dog server 203 processes the compiled dog file 205 and provides a dynamic output response to its communication manager 206. The communication manager 206 of the dog server 203 manages the output response and may provide a response to the web browser's request through the point of entry provided by the web server 202. The web browser 201 interprets the response.

FIG. 3 illustrates an exemplary flowchart for a web browser 201 providing a request to a web server 202 according to an embodiment of the present invention. The exemplary flowchart of FIG. 3 can be implemented, for example, in software that is stored in, for example, memory of a general purpose computer system 101 as shown in

FIG. 1 and that executes on the processor of the general purpose computer 101.

The user's request may be provided to the web browser 201 by entering a page address as shown in step 301, clicking a hypertext link as shown in step 302, submitting an HTML form as shown in step 303, or selecting a bookmark as shown in step 304. Further, a user may be redirected to a new address from which a request to the web browser 201 may be supplied as shown in step 305. The request supplied by the user to the web browser 201 may include a request for a web page, a request for a dog file, and/or for additional information. In step 306, the web browser 201 determines and locates the server that includes, for example, the desired web page. In step 307, the web browser 201 sends a request to the web server, for example, for the web page with the information requested by the user.

FIG. 4 illustrates an exemplary flowchart for a web server 202 processing a request from a web browser 201 according to an embodiment of the present invention. The exemplary flowchart of FIG. 4 can be implemented, for example, in software that is stored in, for example, memory of a general purpose computer system 101 as shown in FIG. 1 and that executes on the processor of the general purpose computer 101.

In step 401, the web server 202 receives the request from the web browser 201. In step 402, the web server 202 determines whether

a dog file is being requested. In an exemplary embodiment of the present invention, the extension of a file is analyzed to determine the file type. In step 403, files which can be interpreted by the web browser 201 such as HMTL files (html extension), text files (txt extension), Graphics Interchange Format files (gif extension), and Joint Photographic Experts Group files (jpg extension), are sent to the web browser 201 from, for example, the web server 202. The request for dog files which can not be interpreted by a web browser 201 (e.g., file having a dog extension), however, is sent to the dog server 203 in step 404.

FIG. 5A illustrates an exemplary flowchart for a dog server 203 processing a request from the web server 202 according to an embodiment of the present invention. The exemplary flowchart of FIG. 5A can be implemented, for example, in software that is stored in, for example, memory of a general purpose computer system 101 as shown in FIG. 1 and that executes on the processor of the general purpose computer 101.

In step 501, the dog server 203 receives the request from the web server 202, for example, for a dog file. The dog server 203 may perform pre-processing as shown in step 502. In an exemplary embodiment of the present invention, pre-processing may include verifying that the requested file exists, for example, on a file system; performing security measures; and allocating resources necessary for the compilation of the dog file and the execution of

the compiled dog file. The security measures performed may include determining that the user making the initial request to the web browser 201 has access to any necessary sources such as a database necessary to complete the request.

Access can be determined based on the additional information provided by the user such as the user identification and password. Allocation of resources may include establishing a communication link with a source such as a database needed to complete the respective compilation and/or execution. In an exemplary embodiment of the present invention, the file system is a resource provided by UNIX platform to access, for example, data storage medium 109 as shown in FIG. 1. If the file requested can not be verified, an indication such as an error message would be provided by the dog server 203. In step 503, the dog server 203 provides an extraction request to the dog kennel 204 for the extraction of a dog file.

FIG. 5B illustrates an exemplary flowchart for a dog kennel 204 processing a dog file extraction request from a dog server 203 according to an embodiment of the present invention. The exemplary flowchart of FIG. 5B can be implemented, for example, in software that is stored in, for example, memory of a general purpose computer system 101 as shown in FIG. 1 and that executes on the processor of the general purpose computer 101.

In step 510, the dog kennel 204 receives the dog file extraction request. In step 511, the dog kennel 204 determines

whether the compiled dog file is in the dog kennel 204. If so, the dog kennel 204 provides a copy of the compiled dog file 205 to the dog server 203 as shown in step 514. If the dog file is not in the dog kennel 204, the dog kennel 204 loads the dog file from a system such as the respective file system at which it may be stored as shown in step 512. In step 513, the dog file is compiled, that is, the execution steps of the compiled dog file are determined and stored as a tree of dog elements (compiled dog file). The compiled dog file 205 is provided to the dog kennel 204 as shown in step 515 and a copy of the compiled dog file is provided to the dog server 203.

In an exemplary embodiment of the present invention, the compilation of a dog file includes the dog kennel 204 identifying components such as raw text, beginning tags and end tags within the dog file. The respective dog elements created are based on the order of the identified components in the dog file. The creation of a dog element includes, for example, identifying the tag name within the beginning tag and determining the package which includes the dog elements from a list of known packages. In an exemplary embodiment of the present invention, the dog kennel creates the list of known packages at initialization. Based on the identified package and tag name, the implementation of the dog element is dynamically created with its execution method.

The tree of dog elements is, for example, an abstract representation of the execution steps of the entire dog file. For example, the execution of the dog file is the sum of the execution of each of the dog elements. In an exemplary embodiment of the present invention, a dog tree element has a constructor which may take a parameter object and an execute method. A dog element may take one of the several forms including a standalone tag, a container tag, an expression and an error element. The error element provides an indication of a compiler error, while an expression is a dog element that is raw text and simply undergoes variable substitution. The execute method of a dog element is the set of instructions that are performed when the element is executed. In the exemplary embodiment of the present invention, the instructions that the execution method may include are any set of instructions that can be programmed with, for example, the JAVA language. A set of instructions may be provided, for example, to interface, alter, and manipulate external systems. Further, the execute method operates on two objects a context and a symbol table.

A context is a group of communication elements used by the execution method of the dog elements. The communication elements are the means by which the execution method interacts with other systems and/or communication manager 206. In an exemplary embodiment of the present invention, the context may include a JDBC connection to a database and an output stream to the communication

manager 206. The symbol table is the environment in which dog element can execute on. The environment serves the purpose of the storage of all variables, e.g., preset and user-defined, in addition to their values. A symbol table includes all the variables that have been set and modified in previous dog element executions. Since the environment is not a fixed entity, it can be used to generate dynamic output using the execute method.

The execution of the dog element also provides a status object that indicates the completion state of the execution, e.g., whether the execution was successful. In an exemplary embodiment of the present invention, the status may be exit (e.g., halt and do no more after the execution of the respective dog element), abort (e.g., stop execution and undo the execution of the other dog elements in the respective compiled dog file 205), and proceed (e.g., continue).

FIG. 6 illustrates an exemplary flowchart for a dog server 203 processing a compiled dog file 205 provided by a dog kennel 204 according to an embodiment of the present invention. The exemplary flowchart of FIG. 6 can be implemented, for example, in software that is stored in, for example, memory of a general purpose computer system 101 as shown in FIG. 1 and that executes on the processor of the general purpose computer 101.

In step 601, the dog server 203 analyzes the compiled dog file. The analysis may include, for example, a determination of the output format, management strategy of the communication, and whether or not

to open communication to either external systems. In the exemplary embodiment of the present invention, the output format may include standard mime types such as html3.2, ASCII, javascript™, and rtf. In step 602, the dog server 203 may open communication to other systems based on analysis of the compiled dog file. In the exemplary embodiment of the present invention, the database to which the dog server 203 establishes communication with may be predetermined and provided in the configuration of the dog server 203. In step 603, the dog server 203 initializes the symbol table, e.g., the environment in which data is stored to be accessed and modified by the dog elements. For example, input variables including preset variables and user-defined variables such as variables passed in from the web browser request.

In step 604, the compiled file is executed. In an exemplary embodiment of the present invention, the execution will include each dog element of the compiled dog file 205 being recursively executed. When execution occurs, the dog server 203 is given the environment to execute on and a way to generate an output response (such as HTML). Each execution of a dog element of the compiled dog file 205 may modify the environment causing executions of other dog elements to execute differently. Accordingly, dynamic information may be generated. Further, the dog tags are dynamically created, that is, the dog tags do not have to be defined at the point of time the dog server 203 is opened. Thus, the dog server 203 does not need to

know all information about the dog tag. Also, the use of the dog tags and execution of the compiled dog tag file allow other systems to be interfaced, manipulated, and altered through the use of the execute method of each dog element. In an exemplary embodiment of the present invention, the other systems may include database systems, email systems and file systems.

In an exemplary embodiment of the present invention, the communication manager 206 of the dog server 203 controls the dynamic output response from the execution of the dog tree elements. The dynamic output response is the set of information that is generated by the execution of the dog element tree, e.g., the compiled dog file. The communication manager 206 determines what format to provide the dynamic output to the web browser 201 based on the output response it receives. For example, if the dynamic output response includes binary information about an image, the communication manager 206 may provide the actual image itself as opposed to HTML that describes the image. The output response may occur during the execution of the compiled dog file or it may be buffered and provided upon completing the execution of the compiled dog file.

In an exemplary embodiment of the present invention, the communication manager 206 of the dog server 203 captures and manages the dynamic output response, determining when the dynamic output should be provided to the web browser 201 through the

point of entry, e.g., portal, provided by the web server 202 as shown in step 606. In an exemplary embodiment of the present invention, the web server 202 does not provide any interpretation or modification of the dynamic output. Thus, the communication manager 206 has the ability to nullify the actions of the execution of a dog element prior to passing the dynamic output to the web browser 201. This may be desired, for example, if the execution of a latter dog element is unsuccessful, although execution of a previous dog element of the same compiled dog file 205 was successful.

After the execution of the compiled dog file, e.g., the tree of dog elements, the dog server 203 may perform post-processing as shown in step 605. In an exemplary embodiment of the present invention, post-processing may include closing established communications, flushing all output streams, and finalizing the request from the web server.

FIG. 7 illustrates an exemplary flowchart for a web browser 201 interpreting a response to its request according to an embodiment of the present invention. The exemplary flowchart of FIG. 7 can be implemented, for example, in software that is stored in, for example, memory of a general purpose computer system 101 as shown in FIG. 1 and that executes on the processor of the general purpose computer 101.

In step 701, the web browser 201 receives the response from the dog server 203 through the web server. In step 702, the web browser 201 determines the format of the response provided by the dog server 203 through the web server 202. In step 703, the web browser 201 displays, for example, on the display device 106 shown in FIG. 1, the information provided by the response it received from the dog server 203 through the web server.

An exemplary embodiment of the present invention includes core dog tags listed in the attached Appendix. An implementer, however, may create custom dog tags. FIG. 8 illustrates an exemplary flowchart for creating a custom dog tag according to an exemplary embodiment of the present invention. In an exemplary embodiment of the present invention, a dog tag is created by providing a tag name as shown in step 801. The tag name must be unique, that is, it may not be one that is already associated with a dog tag. A tag name is the text used by an author to define a tag and the respective parameters. In step 802, the type of dog tag to be created, for example, the first type of tag (a tag that is compiled into a standalone tag) or the second type of tag (a tag that is compiled into a container tag) is determined. If the type of dog tag being created requires parameters, parameters are provided including their respective types as shown in step 803. For example, parameter types may be a literal, expression, number, list of identifiers, single identifier, and empty. An empty type parameter affects the behavior

of the respective tag based on whether a parameter is present, not on the value of the parameter. The dog tag being created is assigned to a tag package as shown in step 804. In an exemplary embodiment of the present invention, it may be either assigned to an existing tag package or a new package which is added to a list of potential tag packages.

In step 805, the class for the type of dog tag being created is extended, which gives the newly created class the same functionality of the class being extended. In the exemplary embodiment of the present invention, extending a class, for example in JAVA, provides a means to identify the type of dog tag while compilation occurs. In step 806, the execute method for the dog tag being created is implemented. In an exemplary embodiment of the present invention, the execute method would include a set of instruction provided by the implementer and written, for example, in the JAVA language. The set of instructions would allow the dog tag to provide a desired function as determined by the implementer such as to interface with other systems. Further, the execute method operates on the context and symbol table. Since the symbol table, e.g., the environment on which the execution method executes, may change prior to the execution method being performed, dynamic output may be generated. For example, the symbol table may have changed as a result of the execution of a prior dog element.

In step 807, a library is imported. The library, for example, com.cpm.dog, may include the objects necessary to implement the basic functionality of the dog tag. For example, the class, context, symbol table, and parameters. In an exemplary embodiment of the present invention, an input/output (I/O) handling package, for example, javaio.IOException is also imported. In step 808, a constructor is provided. In an exemplary embodiment of the present invention, the constructor may include the ability to capture error messages such as those provided by an I/O handling package regarding the creation of the compiled dog file. In step 809, a file is created, such as a JAVA file, implementing the steps of providing a unique tag name as shown in step 801, determining the type of dog tag as shown in step 802, providing parameters and the parameters respective types as shown in step 803, assigning the dog tag to an existing package as shown in step 804, extending the class of the type of dog tag as shown in step 805, implementing the execute method as shown in step 806, importing a library as shown in step 807, and providing a constructor as shown in step 808. In an exemplary embodiment of the present invention, a step of importing an I/O handling package may be implemented by creating the file. In an exemplary embodiment of the present invention the dog tag is compiled as shown in step 810. Further, if the package the dog tag was assigned to, e.g., created in, is not in an existing package, the dog server must be configured with the location of the package

and be restarted as shown in step 811. The compilation provides the file created in step 809 in a form that can be used by the JAVA virtual machine.

The embodiments described above are illustrative examples of the present invention and it should not be construed that the present invention is limited to these particular embodiments. Various changes and modifications may be effected by one skilled in the art without departing from the spirit or scope of the invention as defined in the appended claims.

APPENDIX

Author Reference

Variable Tags

<!SET>

This standalone tag sets the contents of one or more user variables. Use the name of the variable you want to set as the parameter name, and the expression you want to set it to as its parameter value.

Format:

<!SET varname=varvalue>

Parameters:

APPEND (empty)
If present, the variable being set in this tag will be treated as a list. The value assigned to it will be added to the end of the list, rather than replacing the whole pre-existing value.

DEFAULT (empty)
If present, SET will not replace any pre-existing value held by the variable. That is, it will perform its duty on the named variable only if it is currently blank.

REPARSE (empty)
If present, after SET has finished evaluating an expression in preparation for assigning it to a variable, it will evaluate it a second time.

VARTYPE (literal choice)
Optional, default is "text". Value must be "text", "list", or "file" (case insensitive). If VARTYPE=LIST is specified, SET will read the given expression as a comma-separated list and will create a list variable containing the items in that list. For VARTYPE=FILE, SET will look for a file with the given name on the server and create a file variable if it is found.

Examples:

<!SET name=Jack>
<!SET greeting="Hello, {name}!">

Now the variable greeting contains "Hello, Jack!".

Any existing value held by the variable will be overwritten in favor of its new value, even if the new value is blank. The following three statements all have the effect of clearing the contents of the foo variable:

<!SET foo="">
<!SET foo=>
<!SET foo>

Another example:

<!SET foo=bar>
<!SET foo=bat APPEND>

Now foo is a list variable containing "bar" and "bat" as elements.

<!SET foo=>

(now foo is empty)
<!SET foo=bar DEFAULT>
(now foo is set to "bar")
<!SET foo=bat DEFAULT>
(foo is still set to "bar")
<!SET foo=bat>
(now foo is set to "bat")

<!SET age=42>
<!SET phrase="You are {lbrace}age{rbrace} years old.">

The variable phrase now contains "You are {age} years old.". In order to have "42" substituted for "{age}", we must reparse the expression:

<!SET phrase="You are {lbrace}age{rbrace} years old." REPARSE>

After the first evaluation, the value of phrase would be, "You are {age} years old". This expression is then evaluated again, so that phrase now contains, "You are 42 years old".

<!SET foo="foo, bar, baz">
(Now foo is literally "foo, bar, baz")

<!SET foo="foo, bar, baz" VARTYPE=LIST>
(foo is now a list containing three elements: "foo", "bar", and "baz")

NOTE: If an equals sign is found after an APPEND, DEFAULT, or REPARSE tag, it will be treated as a variable name to be set instead of affecting the overall behavior of SET. If VARTYPE has any value other than "text," "list," or "file," it should also be treated as a variable to be set.

SET does have a single anomaly: you cannot use it to directly set a variable named "vartype" to the value "text", "list", or "file".
To do so, use a construct such as:

<!SET tmp=text>
<!SET vartype="{tmp}">
<!SET tmp="">

You can actually specify as many variable identifiers as you wish as parameters to SET. Each new value that is specified is treated as an expression (see "Parameter Types") which may contain variables itself. This expression will be evaluated and the results of the evaluation will be stored in the new variable. Any additional parameters you have specified (DEFAULT, VARTYPE, etc.) will apply to all variables being set within the tag. Caution: The order in which the variables will be set is not defined.

<!SETVAR>

This standalone tag performs the same basic function as SET in a different manner which is more suitable for some situations.

Parameters:

NAME (expression)
Required. The variable name that should be set.

VALUE (expression)
Required. The value that should be assigned to the named variable.

Examples:

```
<QUERY SQL="
  SELECT Var_Name, Var_Value
  FROM  Data_Table
">
<EACHROW>
    <SETVAR NAME="{var_name}" VALUE="{var_value}">
<EACHROW>
<QUERY>
```

<CLEAR>

This standalone tag undefines a set of variables.

Parameters

VARS (list of identifiers)
Required. The variable names whose contents should be cleared (that is, set to "").

Examples:

```
<!CLEAR VARS=name,address,city,state,zip>
```

<!EACHVAR>

EACHVAR is a container tag that loops through a series of variables determined by the parameters given. Regardless of the parameters, though, the behavior each time through the loop is the same. It will set the reserved variable dog.name to the current variable name and dog.value to the current variable value. These variables are cleared when the loop finishes.

Parameters:

ALL (empty)
If present, EACHVAR loops through all user and reserved variables. Otherwise, it only loops through all user variables.

PREFIX (expression)
Optional. If specified, the expression is evaluated and then EACHVAR loops through all variables that begin with that sequence of letters. When PREFIX is specified, both user variables and reserved variables can be accessed just the same, depending only on the prefix you specify.

Examples:

To loop through the Java system properties:

```
<UL>
<!EACHVAR PREFIX=sys.>
  <LI>{dog.name} = {dog.value}
```

```
</!EACHVAR>
</UL>
```

<!EACHITEM>

EACHITEM, like EACHVAR, is a container tag that results in a loop. Instead of looping through all variables, though, it loops through all elements in one particular list variable. Inside the EACHITEM block, the current element of the list can be accessed in the same variable that references the whole list outside the block.

Parameters:

LIST (identifier)
Required. The list variable to examine.

Examples:

```
<!SET foo="foo, bar, baz" VARTYPE=list>
{foo}<BR>
<!EACHITEM LIST=foo>
   {foo}<BR>
</!EACHITEM>
{foo}<BR>
```

produces the following output:

```
[foo, bar, baz]<BR>
   foo<BR>
   bar<BR>
   baz<BR>
[foo, bar, baz]<BR>
```

Remember the LIST parameter takes an identifier, not an expression, so curly braces are not used.

Conditional Tags

<!IFDEF>

IFDEF is a container tag that causes its block to be executed only if a given variable or variables are nonempty.

Parameters:

IFDEF has no parameters other than the variable names you wish it to check. Any number of variables can be specified. The statements enclosed by IFDEF will be executed if at least one of the identifiers specified is defined.

Examples:

```
<!SET foo=bar>
<!IFDEF foo>
   Hello
</!IFDEF>
```

```
<!SET foo=>
<!IFDEF foo>
  Goodbye
</!IFDEF>
```

This will display "Hello" but not "Goodbye".

```
<!SET a="" b="" c=1>
<!IFDEF a b c>
  Yes!
</!IFDEF>
```

This will produce the output "Yes!" IFDEF implicitly performs an "OR" between the named variables: "If a is set OR b is set OR c is set." Note that an equivalent "AND" capability is not needed because IFDEF tags can nest inside each other like this:

```
<!IFDEF a>
  <!IFDEF b>
    Yes, both a and b are set!
  </!IFDEF>
</!IFDEF>
```

An error results if IFDEF is used without naming any variables.

<!IFNDEF>

IFNDEF is a container tag that performs the exact opposite function of IFDEF. The statements it encloses are executed if and only if the variable named is undefined, that is, has a value of "".

Parameters:

IFNDEF has no parameters other than the variable names you wish it to check. Any number of variables can be specified. The statements enclosed by IFNDEF will be executed if at least one of the identifiers specified is undefined.

Just like IFDEF, when multiple variables are specified, IFNDEF performs an OR: "If a is not set OR b is not set OR c is not set." This ensures a specific set of values has been filled in.

Examples:

```
<!IFNDEF name age address city>
  Error! You must fill in all the required fields.
</!IFNDEF>
```

An error results if IFNDEF is used without naming any variables.

<!IFEQ>

IFEQ is a container tag that works very similarly to IFDEF and IFNDEF. The simplest use of IFEQ is to test whether two variables have the same value. The names of these two variables are given as parameters,

and IFEQ will cause the enclosed statements to be executed only if the contents of those variables are the same.

Parameters:

IFEQ has no parameters other than the variable names you wish it to check. Any number of variables can be specified. The statements enclosed by IFEQ will be executed if all the named variables are equal to each other.

Examples:

```
<!SET foo="A phrase">
<!SET bar="Another phrase">
<!SET baz="Another phrase">
<!IFEQ foo bar>
  These words will not appear.
</!IFEQ>
<!IFEQ bar baz>
  These words will appear!
</!IFEQ>
```

You can specify more than two variables. The enclosed statements are executed if and only if all the named variables are equal to each other.

<!IFEQ foo bar baz> These words will not appear. </!IFEQ>

An error results if IFEQ is used without naming at least two variables.

<!IFIN>

This container tag executes its block if and only if a given element is found in the named list variable.

Parameters:

LIST (identifier)
Required. The name of the list-type variable to examine.

ELEMENT (expression)
Required. After this expression is evaluated, the resulting text will be searched for in the named list variable.

Examples:

```
<!SET alist="foo, bar, baz" VARTYPE=LIST>
<!IFIN LIST=alist ELEMENT=bar>
    This sentence will appear in the output.
</!IFIN>
<!SET atext=ba>
<!IFIN LIST=alist ELEMENT="{atext}z">
    As will this one.
</!IFIN>
<!IFIN LIST=alist ELEMENT=test>
    But not this one.
</!IFIN>
```

<!SWITCH>
<!CASE>
<!DEFAULT>

SWITCH is a container tag inside which any number of standalone CASE tags may appear, optionally followed by one standalone DEFAULT tag. CASE and DEFAULT may appear only immediately inside a SWITCH tag and may not be contained by any other container tag.

Format:

```
<!SWITCH VALUE="{somevar}">
<!CASE VALUE=firstliteral>
  ... statements ...
<!CASE VALUE=secondliteral>
  ... statements ...
<!CASE VALUE=thirdliteral>
  ... statements ...
<!DEFAULT>
  ... statements ...
</!SWITCH>
```

Parameters:

(SWITCH tag)

VALUE (expression)
Required. Evaluated immediately. The resulting value will then be compared, one by one, to the VALUE parameter of
each enclosed CASE tag.

(CASE tag)

VALUE (literal)
Required. If this literal matches the SWITCH value, the statements following this CASE tag will be executed until the next , , or tag is reached.

(DEFAULT tag)

No parameters. If no matching CASE was found, the statements following this tag and preceding the end of the SWITCH block will be executed.

Examples:

```
The number you have entered is
<!SWITCH VALUE="{number}">
  <!CASE VALUE=1> one.
  <!CASE VALUE=2> two.
  <!CASE VALUE=3> three.
  <!CASE VALUE=4> four.
    ...
  <!CASE VALUE=20> twenty.
  <!DEFAULT> not a whole number between 1 and 20!
</!SWITCH>
```

One common use of the SWITCH tag is to consolidate awkward statements like:

```
<!IFNDEF foo>
  Do this
</!IFNDEF>
<!IFDEF foo>
  Do that
</!IFDEF>
```

into:

```
<!SWITCH VALUE="{foo}">
  <!CASE VALUE=>Do this
  <!DEFAULT>Do that
</!SWITCH>
```

<!RANDOM>
<!ITEM>

The RANDOM container tag and ITEM standalone tag allow you to have statements randomly selected from a group to be executed. RANDOM works much like the SWITCH tag, except there is no DEFAULT option, and the block of statements to be executed is up to chance instead of the value of an expression. ITEM can appear only immediately inside a RANDOM tag. If no ITEM tags are in a RANDOM block, or all the ITEM tags there have WEIGHT=0, RANDOM will produce an error.

Parameters:

WEIGHT (number)
Optional, default "1". Allows you to configure certain choices to be more likely to be chosen than others. It is legal to specify WEIGHT=0, but then the statements following that ITEM tag would never be chosen.

Examples:

```
You are rolling a six-sided die.  You rolled a
<!RANDOM>
  <!ITEM>one!
  <!ITEM>two!
  <!ITEM>three!
  <!ITEM>four!
  <!ITEM>five!
  <!ITEM>six!
</!RANDOM>

Now you're rolling two four-sided dice.  You rolled a
<!RANDOM>
  <!ITEM WEIGHT=1>two!
  <!ITEM WEIGHT=2>three!
  <!ITEM WEIGHT=3>four!
  <!ITEM WEIGHT=4>five!
  <!ITEM WEIGHT=3>six!
  <!ITEM WEIGHT=2>seven!
  <!ITEM WEIGHT=1>eight!
```

```
</!RANDOM>
```

You could use RANDOM to implement a sort of "sweepstakes" whereby a few lucky visitors to your site win a prize. This example gives a 1 in 10,000 chance of winning.

```
<!RANDOM>
<!ITEM>
  <P>You have just won a free barbecue set!  Please click here
    to fill out your name and address and we will send it to you!</P>
<!ITEM WEIGHT=9999>
</!RANDOM>
```

File Tags

<!INCLUDE>

INCLUDE is a standalone tag that allows you to include the contents of another document inside the current document.

Parameters:

FILE (expression)
Required. Specifies the document to be included. Either an absolute pathname or a pathname relative to the current document can be given. URL translation rules will not apply. If the filename ends in .dog, the included file will be executed by the DOGtags parser; otherwise the contents of the file will be copied to the output as-is.

VARS (list of identifiers)
Optional. Only meaningful when the FILE parameter references another DOGtags document (that is, the filename ends in .dog). If specified, only copies of the variables you name will be passed into the included file. If the included file changes or unsets these values, or defines new variables of its own, these changes will not be reflected in the main file. (All reserved variables accessible in the main file can be accessed in the included file as well.) If VARS is absent, however, the included file will share the same variable memory as the main file. Variables can be set, unset, and modified inside the included file and these changes WILL carry over to the remainder of the main file. If the FILE specified does not end in.dog, the VARS parameter will be ignored.

CANEXIT (empty)
If present, and the included file executes an EXIT tag, the including file will exit as well. By default, EXIT tags inside included files cannot exit the including file.

Examples:

<!LINK>

LINK is a standalone tag that provides a simple mechanism for creating a hyperlink to another DOGtags page.

Parameters:

TEXT (expression)
Optional. The text that should appear on the screen for the user to click on. TEXT can be specified with or without IMG, but if neither is specified, there will be no way for the user to activate the link.

IMG (expression)
Optional. The absolute or relative URL to an image file for the user to click on. IMG can be specified with or without TEXT, but if neither is specified, there will be no way for the user to activate the link.

URL (expression)
Optional. The absolute or relative URL to the DOGpage you are linking to. This link should not contain a query string. If it is not present, an inactive hyperlink will be created; this is a good way to take advantage of the STATUS feature below even if there is no actual hyperlink involved.

VARS (list of identifiers)
Optional. A comma-separated list of the variable identifiers you wish to pass into the next document. If it is not present, no variables will be passed.

COLOR (expression)
Optional. The color you want the link to appear, either as an accepted color name or a six-digit hexadecimal code.

STATUS (expression)
Optional. The phrase you wish to appear in the status bar of the Web browser when the mouse cursor is over the link.

TARGET (expression)
Optional. Specifies the window that the link should be opened in when clicked.

ACTIVE (expression)
Optional. If this parameter is present but the expression evaluates to a blank string, this link will be made unclickable.

ONCLICK (expression)
Optional. Specifies javascript code to be invoked when this link is clicked on.

SIZE (literal)
Optional. The width and height of the named IMG, in the form wwxhh.

Examples:

To pass the variables client and job to another page called details.dog, using the default link color, using the concatenation of client and job as the link text, and displaying the job name in the status bar:

```
<!LINK URL=details.dog VARS=client,job TEXT="{client}{job}"
    STATUS="{jobname}">
```

<!REDIRECT>

REDIRECT is a standalone tag that causes the Web browser to immediately display a new URL in place of the current page.
Any output which has already been produced in the current page will be discarded in favor of the new page.

Parameters:

URL (expression)
Required. A relative or absolute URL to the new page. If URL is not present, or specifies an obviously invalid location, the REDIRECT tag will fail with an error message.

VARS (list of identifiers)
Optional. A comma-separated list of the variable identifiers you wish to pass into the next document. If it is not present, no variables will be passed.

ALLVARS (empty)
If present, it is equivalent to listing all the currently set variables in a VARS parameter. All set variables will be passed to the new page.

Examples:

```
<!REDIRECT URL=http://www.cpm.com/foo.dog VARS=bar,baz>
If this message shows up, there must have been an error
    with the redirect!
```

<!DOWNLOAD>

This standalone tag aborts the current page and causes the Web browser to begin downloading the named file. Any output which has already been produced in the current page will be discarded in favor of the new page.

Parameters:

FILE (expression)
Required. A relative URL to the file to be downloaded.

TYPE (expression)
Optional, defaults to "application/octet-stream". The MIME type of the file.

Examples:

```
<!IFDEF readytodownload>
    <!DOWNLOAD FILE=somefile.zip TYPE=application/zip>
</!IFDEF>
```

<!EACHFILE>

This container tag loops through all of the files present in a given directory, each time setting the variable dog.filename to the current filename.

Parameters:

DIR (expression)
Optional, defaults to ".". The directory whose contents you wish to loop through.

Examples:

<!MOVE>

This standalone tag moves (or renames) a file on the server file system.

Parameters:

FILEVAR (identifier)
Required. A DOGtags File-type variable representing the file to be renamed.

TO (expression)
Required. The new filename for the file.

Examples:

<!MKDIR>

This standalone tag creates a new directory on the server filesystem.

Parameters:

DIR (expression)
Required. The name of the directory you want to create.

Examples:

Database Access Tags
<!FETCH>

This standalone tag executes a query in the database which is expected to return only one row and places the retrieved values into DOGtags variables.

Parameters:

SQL (expression): Required. The SQL SELECT statement that should be issued to the database. Identical to QUERY's SQL parameter.

INTO (list of identifiers): Optional. By default, FETCH stores each returned column value in a variable with the same name as the column name in the query. To use different variable identifiers, specify them here in respective order to the columns selected in the query. For each column returned that does not have a corresponding identifier specified in INTO, the query-based identifier will be used as usual.

INSIST (empty): Ordinarily, if the requested row is not found in the query, FETCH will terminate normally without setting any variables or raising any errors. To indicate that a statement should always successfully return a row, and that it would be an error condition if it did not, add the INSIST parameter to the FETCH tag. If it is present, and the statement does not return any rows, FETCH will abort the current transaction (if applicable) and set a "No rows selected" error.

Examples:

<!FETCH SQL="SELECT Full_Name name, Age FROM People

WHERE Id = 2048">

Hello, {name}! You are {age} years old!

```
<!FETCH SQL="
SELECT Cat_No, (Fig1 + Fig2) / 2 Fig_Avg
FROM Foo
WHERE Id = 2404
">
The average for {cat_no} is {fig_avg}.

<!FETCH SQL="SELECT Name, Age, Hire_Date FROM Emp WHERE EmpNo = 413"
INTO="empname,empage,emphire">
<!-- sets empname, empage, and emphire -->

<!FETCH SQL="SELECT Name, Age, Hire_Date FROM Emp
WHERE EmpNo = 413"
INTO="empname,, emphire">
<!-- sets empname, age, and emphire -->

<!FETCH SQL="SELECT Name, Age, Hire_Date FROM Emp
WHERE EmpNo = 413" INTO=empname,empage>
<!-- sets empname, empage, and hire_date -->

<!FETCH SQL="SELECT Name, Age, Hire_Date FROM Emp WHERE EmpNo = 413"
INTO=",,,foo">
<!-- sets name, age, and hire_date -->

<!QUERY>
<!EACHROW>
<!NOROWS>
```

QUERY and EACHROW are a pair of container tags that function similarly to FETCH, except that they can handle multiple rows being returned from the query. The contents of the EACHROW tag are executed

once for each row returned from the query. EACHROW may appear only immediately inside a QUERY tag and may not be contained by any other container tag.

EACHROW has no parameters. The SQL and INTO parameters of QUERY are identical to those in FETCH. QUERY does not accept the INSIST parameter. Instead, if no rows are found, any statements inside an included NOROWS container tag will be executed.

Parameters:

SQL (expression): Required. The SQL SELECT statement that should be issued to the database. Identical to FETCH.

INTO (list of identifiers): Optional. By default, FETCH stores each returned column value in a variable with the same name as the column name in the query. To use different variable identifiers, specify them here in respective order to the columns selected in the query. For each column returned that does not have a corresponding identifier specified in INTO, the query-based identifier will be used as usual. Identical to FETCH.

SECTIONS (number): The number of sections the query results should be broken into. The entire body of the QUERY tag is executed once for each of these sections. The EACHROW tag inside will run only until the current section is finished and then will finish, only to run again during the next section.

WATCH (list of identifiers): This is a parameter to EACHROW, not QUERY. It specifies the variables names that EACHROW should watch to determine whether they have changed since the previous row. When a new value is found, it will be stored in the variable "dog.new.variablename". (This paragraph needs improvement.)

Examples: (The numbers 1 through 5 are stored in a table.)

```
<!QUERY SQL="SELECT Num FROM Nums ORDER BY Num">

<UL>

<!EACHROW>

<LI>{num}

</!EACHROW>

</UL>

</!QUERY>
```

This DOG syntax produces the following output:

```
<UL>

<LI>1

<LI>2

<LI>3

<LI>4

<LI>5
```

```
</UL>
<!QUERY SQL="SELECT Num FROM Nums ORDER BY Num" SECTIONS=3>
<UL>
<!EACHROW>
<LI>{num}
</!EACHROW>
</UL>
</!QUERY>
```

The output then becomes:

```
<UL>
<LI>1
<LI>2
</UL>
<UL>
<LI>3
<LI>4
</UL>
<UL>
<LI>5
</UL>
```

<!UPDATE>

This standalone tag performs an SQL INSERT, UPDATE, or DELETE statement. Following the tags, the number of rows successfully updated is stored in the dog.rowsUpdated variable. If an error occurs, the appropriate error variables will be set, and dog.rowsupdated will be unset—not simply set to zero.

Parameters:

SQL (expression): Required. The SQL DML or DDL statement to execute. Unlike FETCH and QUERY, it may contain unbound values represented as question marks (?). These values will be bound using the FROM parameter.

FROM (list of identifiers): Optional. Specifies the DOGtags variables whose values should be bound to the question marks (?) in the SQL query. There must be the same number of identifiers listed in the FROM parameter as there are question marks in the SQL parameter, or a database error will result.

ROWS (number): Optional. Specifies the number of rows this UPDATE is expected to act on. If the actual number turns out to be different from the one given here, an error will occur.

Example:

```
<!UPDATE SQL="
UPDATE Accounts
SET Balance = Balance - ?
WHERE Account_No = ?
" FROM=amount,from_account>
```

<!TRANSACT>

This container tag encloses a series of commands which function as a single transaction. If any database tag (FETCH, EVAL, QUERY, UPDATE) inside the TRANSACT block fails, the entire block is exited and all database actions performed since the beginning of the block are undone. Because of this behavior, it is useful to put other tags, such as EMAIL, inside the block so they will be executed only if all the database commands were successful.

TRANSACT tags do not nest, either directly or indirectly.

Parameters:

none

Example:

```
<!TRANSACT>
<!UPDATE SQL="
UPDATE Accounts
SET Balance = Balance - ?
WHERE Account_No = ?
" FROM=amount,from_account>
<!UPDATE SQL="
UPDATE Accounts
SET Balance = Balance + ?
WHERE Account_No = ?
```

" FROM=amount,to_account>

</!TRANSACT>

<!EVAL>

EVAL is the simplest database access tag. It is a standalone tag that evaluates a single expression using the database's SQL engine and either stores the result into a variable or displays it on the output.

Parameters:

EXPR (expression): Required. A valid SQL expression to be evaluated. This is an SQL expression, not a statement, and so cannot contain "SELECT," "FROM," or "WHERE," nor can it reference tables or columns. It can, however, reference "pseudo-columns" such as User and NextVal in Oracle. If EXPR does not evaluate to a valid SQL expression, EVAL will abort the current transaction and set the appropriate error variables.

INTO (identifier): Optional. The DOGtags variable identifier into which the result should be stored. If this parameter is absent, the result will be displayed on the output instead.

Examples:

One plus one is <!EVAL EXPR="1 + 1">.

<!EVAL EXPR="{count} + 1" INTO=count>

<!EVAL EXPR=My_Sequence.NextVal INTO=current_id>

You have been assigned an ID number of {current_id}

That phrase is <!EVAL EXPR="LENGTH({phrase:tosql})"> characters long.

You are logged in to the database as <!EVAL EXPR=LOWER(User)>

Form tags

<!FORM>

FORM is a container tag that functions similiarly to the HTML FORM tag but with more features.

Parameters:

URL (expression)
Optional. The absolute or relative URL to the DOGpage that should be requested when this FORM is submitted.

METHOD (literal "GET", "POST", "FILE")
Optional, defaults to "GET". The method the form should be submitted with, either "GET", "POST", or "FILE" (which is like POST but allows for file upload components).

TABLE (expression)
Optional. Causes the field sizes of the named table in the database to be loaded into memory to serve as default MAXLENGTHs for any TEXTFIELD tags within this FORM.

SAVEDIR (expression)
Optional, defaults to "/tmp/dog". Valid only for METHOD=FILE; this specifies the directory into which any files uploaded via this form should be saved. It can be specified with an absolute pathname or a relative pathname to the current document.

Examples:

<!SEND>

This standalone tag sends the specified variables to the next page (as hidden fields).

Parameters:

VARS (list of identifiers)
Required. Specifies the DOGtags variables that should be passed along to the next page (as hidden fields).

Examples:

<!TEXTFIELD>

Description

Parameters:

VAR (identifier)
Required. The name of the DOGtags variable that this textfield will set. Any current value held by this variable will be automatically filled in to the textfield.

MAXLENGTH (number)
Optional. The maximum number of characters the user can enter into this textfield. If TABLE was specified in this FORM's parameter list, and the VAR parameter of this TEXTFIELD matches a column name in that table, and MAXLENGTH is missing or zero, it will default to that column's field width.

SIZE (number)
Optional. Specifies the on-screen size of the textfield. If size begins with a '+' or '-' character, it will be interpreted as relative to MAXLENGTH.

NOECHO (empty)
If present, characters in this textfield will appear as *'s.

Examples:

<!CHOICE>

Description

Parameters:

TYPE (literal "PULLDOWN", "RADIO", "RADIOLINE")

Optional, defaults to "PULLDOWN". The display style for this CHOICE component. PULLDOWN = a pull-down box; RADIO = a series of radio buttons one on top of the other; RADIOLINE = a series of radio buttons all on one line.

VAR (identifier)
Required. The name of the DOGtags variable that this choice component will set. Any current value held by this variable will be automatically selected in the choice component.

TABLE (literal)
Required. The name of the table containing the valid choices for the user to pick from in this choice component.

VALUES (literal)
Optional, defaults to the value of VAR. The column (or column expression) in TABLE containing the legal values that can be assigned to the variable VAR.

DISPLAY (expression)
Optional, defaults to the value of VALUES. The column (or column expression) in TABLE containing the text that should be displayed in the on-screen choice component for each choice.

WHERE (expression)
Required if MORE is present. The SQL "where" clause that should be used to restrict the number of choices offered to the user. If not present, all rows will be returned.

MORE (empty)
If present, then following the set of choices specified by the WHERE parameter will appear a choice labeled "More choices...". If the user selects it, the choice component will reload without applying the WHERE parameter.

ORDER (expression)
Optional. The SQL "order by" clause that should be applied to these choices.

MANDATORY (empty)
The presence of this parameter identifies this choice component as one for which the user must make a choice. Thus if the variable has a previous value (whether through retrieval from the database, or simply a default value), there will be no "Select one" or "None" option in the choice component; the only clickable items will be valid choices.

BLANK (expression)
Optional, defaults to "Select one" or "None" (depending on MANDATORY). This is the wording that should appear on the initial "blank" choice. For non-mandatory choice components, this will always appear at the beginning of the list of choices and defaults to "None". For mandatory choice components, this will only appear when there is no previous (or default) value for the variable, and the phrasing defaults to "Select one".

Examples:

Other Tags

<!DOGTYPE>

This standalone tag must be the very first object in the file. It configures the output type that this DOGfile will produce.

Parameters:

OUTPUT (literal)
Required. The type of output that this DOGfile produces. This value is case insensitive and can currently be set to any of the following values: ascii, html, html3.2, html4.0, javascript, jscript, or rtf.

Examples:

To begin the typical DOGpage, which produces HTML output:

<!DOGTYPE OUTPUT=html3.2>

To begin a DOGpage designed to generate javascript code:

<!DOGTYPE OUTPUT=javascript>

<!EMAIL>

This container tag sends an e-mail message. The recipients, sender, and subject line are configured as parameters to the start tag, and all the text between the start and end tags is sent as the body of the message.

Parameters:

TO (expression)
Required. The recipient(s) of the message.

CC (expression)
Optional. Any carbon-copy recipient(s) of the message.

FROM (expression)
Optional. The apparent sender of the message. If not specified, or if the result of evaluating the expression is empty, a site-configured default should be used.

SUBJECT (expression)
Optional. Subject line for the message.

Examples:

```
<!EMAIL FROM="{my_email}" TO=register@foo.com
  SUBJECT="Event registration from {fullname} received">
Hello,

The following registration form was submitted:

  Name:   {fullname}
  Address: {street}
       {city}, {state} {zip}
  Phone #: {telephone}

<!IFDEF my_email>
You can contact this person at {my_email}.
</!IFDEF>
</!EMAIL>
```

<!MATCH>

This standalone tag performs a Perl5-compatible regular expression search on a named Text-type variable.

Parameters:

VAR (identifier)
Required. The name of the DOGtags variable to check.

REGEX (literal)
Required. A regular expression (in Perl5-compatible syntax) to look for in the named variable.

INTO (list of identifiers)
Required. The DOGtags variables into which saved groups from the regular expression match should be stored.

NOCASE (empty)
If present, differences in cases between the pattern and the text of the variable being searched are ignored.

Examples:

<!INVOKE>

This standalone tag invokes a Java method. The method you wish to invoke must accept a single parameter, a String array, and must return a String array as well. This is important even if your method has no real input or output. The method and the class it is in must be public, and the method must be static.The method can throw any exceptions you wish; DOGtags will display the exception message in the output.

Java Method Format:

```
public static String[] myMethod(String[] args) throws Exception
```

Parameters:

METHOD (literal)
Required. The fully qualified method name, expressed as package.class.method. For example, com.cpm.util.MyClass.myMethod.

FROM (list of identifiers)
Optional. The variables whose values should form the String array passed into the method. If not present, an empty array will be passed.

INTO (list of identifiers)
Optional. The variables whose values should be set from the String array returned by the method. If not present, the returned array is ignored.

Examples:

```
package com.cpm.util;

import java.util.Random;

public class RandomInteger {
```

```
    private static Random numGen = new Random();

    public static String[] generate(String[] args) throws Exception {

      int min = Integer.parseInt(args[0]);
      int max = Integer.parseInt(args[1]);
      if (max < min)
        throw new Exception("max < min");

      int range = max - min + 1;
      int result = min + (int)(range * numGen.nextDouble());

      String[] tmp = { new String(result) };
      return tmp;
    }
  }
```

And here is the DOGtags syntax to invoke that method:

```
<!SET min=1002 max=9998>
<!INVOKE METHOD=com.cpm.util.RandomInteger.generate
  FROM=min,max INTO=pin>
You have been assigned a PIN of {pin}.
```

<!COMMENT>

This container tag compiles the enclosed statements, but does not execute them.

Parameters:

none

Examples:

```
<!COMMENT> Now we make the database updates... </!COMMENT>
```

<!EXIT>

This standalone tag causes execution of the current page to terminate. It is most useful inside a conditional tag of some kind; otherwise, the statements following it would never be executed under any circumstances.

Parameters:

none

Examples:

```
<!IFNDEF name>
  Error! You must enter your name!
  <!EXIT>
</!IFNDEF>
```

What is claimed is:

1. A method for generating dynamic output from a system, the method comprising the following steps:

a web browser providing a web browser request to a web server;

the web server determining whether a dog file is being requested by the web browser request, the dog file including a text file having a predetermined syntax;

if the dog file is not being requested, the web server providing to the web browser a response to the web browser request;

if the dog file is being requested, the web server providing a web server request to a dog server, wherein the dog server provides a dog file extraction request to a dog kennel to extract a compiled dog file from the dog kennel, the dog kennel providing a copy of the compiled dog file to the dog server, the dog server executing the compiled dog file and providing the dynamic output to the web browser.

2. The method according to claim 1, wherein a dog file is a file that includes components that can be compiled into specific implementations of a requirement that an execute method that operates on a context and a symbol table while returning a status exists.

3. The method according to claim 1, wherein the step of the web server determining whether the dog file is being requested by the web browser includes analyzing a file extension of a file being requested by the web browser.

4. The method according to claim 1, further comprising the following steps:

the dog kennel determining whether the compiled dog file is in the dog kennel; and

if the compiled dog file is not in the dog kennel, the dog kennel loading the dog file from a file system, compiling the dog file, storing the compiled dog file in the kennel, and providing a copy of the compiled dog file to the dog server.

5. The method according to claim 1, further comprising the following steps:

the dog server analyzing the compiled dog file;

the dog server establishing communication with the system; and

the dog server setting up an environment for the execution of the compiled dog file.

6. The method according to claim 1, wherein the compiled dog file includes a plurality of dog elements, each of the plurality of dog elements includes an execute method, wherein the execute

method returns a status object and operates on a context and a symbol table to produce the dynamic output.

7. The method according to claim 1, wherein the execution of the compiled dog file includes providing a dynamic output response to a communication manager of the dog server, wherein the communication manager manages the dynamic output response provided as a result of the compiled dog file.

8. The method according to claim 1, wherein the dynamic output is provided to the web browser through a portal provided by the web server.

9. An apparatus for generating dynamic output from a system, the apparatus comprising:

a web browser;

a web server coupled to a web browser;

a dog server coupled to a web server; and

a dog kennel coupled to a dog server, wherein the web browser provides a web browser request to the web server, the web server determines whether a dog file is being requested by the web browser request, if the dog file is not being requested, the web server provides to the web browser a response to the web browser request; if the dog file is being requested, the web server provides a web server request to the dog server based on the web browser request, the dog server provides a dog file

extraction request to the dog kennel, the dog kennel provides a copy of a compiled dog file to the dog server, and the dog server executes the copy of the compiled dog file and provides the dynamic output to the web browser.

10. The apparatus according to claim 9, wherein the dynamic output is provided to the web browser through a portal provided by the web server.

11. The apparatus according to claim 9, wherein the dog server includes a communication manager.

12. The apparatus according to claim 9, wherein the execution of the compiled dog file includes providing an output response on manager of the dog server, wherein the communication manager manages the dynamic output response provided as a result of the compiled dog file.

13. A method for creating a dog tag which is associated with a class and a type, the method comprising the following steps:

providing a unique tag name for the dog tag;

determining the type of the dog tag;

assigning the dog tag to a package;

extending the class for the type of the dog tag;

implementing an execute method for the dog tag;

importing a library;

providing a constructor; and

creating a file implementing the steps of providing a unique tag name, determining the type of the dog tag, assigning the dog tag to the package, extending the class of the type of the dog tag, implementing the execute method, importing a library, and providing the constructor.

14. The method according to claim 13, further comprising the step of providing at least one parameter and the respective parameter type for the dog tag.

15. The method according to claim 13, wherein the file created is a JAVA file.

16. The method according to claim 13, further comprising the following steps:

compiling the file; and

restarting a dog server.

17. The method according to claim 13, further comprising the following step:

importing an input/output handling package.

18. The method according to claim 13, wherein the library includes objects necessary to implement basic functionality of the dog tags.

19. The method according to claim 13, wherein the step of assigning the dog tag to a package includes assigning the dog tag

to at least one of an existing tag package and a new package which is added to a list of potential tag packages.

20. A method for establishing an interface with an external system, the method comprising the following steps:

a web browser providing a web browser request to a web server;

the web server determining whether a dog file is being requested by the web browser request;

if the dog file is not being requested, the web server providing to the web browser a response to the web browser request;

if the dog file is being requested, the web server providing a web server request to a dog server based on the web browser request, wherein the dog server provides a dog file extraction request to a dog kennel for the dog file requested by the web browser, the dog kennel providing a copy of a compiled dog file to the dog server, and the dog server executing the compiled dog file, wherein the execution of the compiled dog file establishes an interface with the external system.

FIG. 1

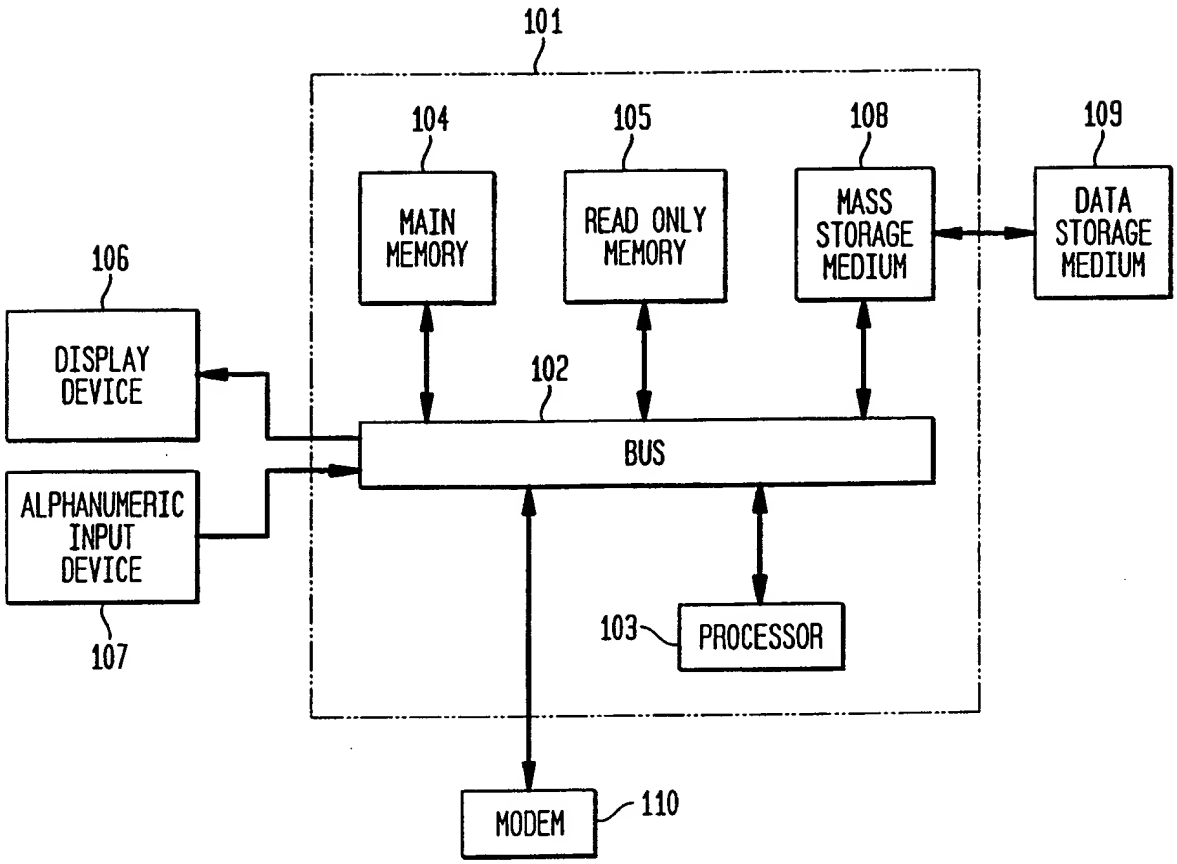

101
104
105
108
109
MAIN MEMORY
READ ONLY MEMORY
MASS STORAGE MEDIUM
DATA STORAGE MEDIUM
106
DISPLAY DEVICE
102
BUS
ALPHANUMERIC INPUT DEVICE
107
103
PROCESSOR
MODEM
110

FIG. 2

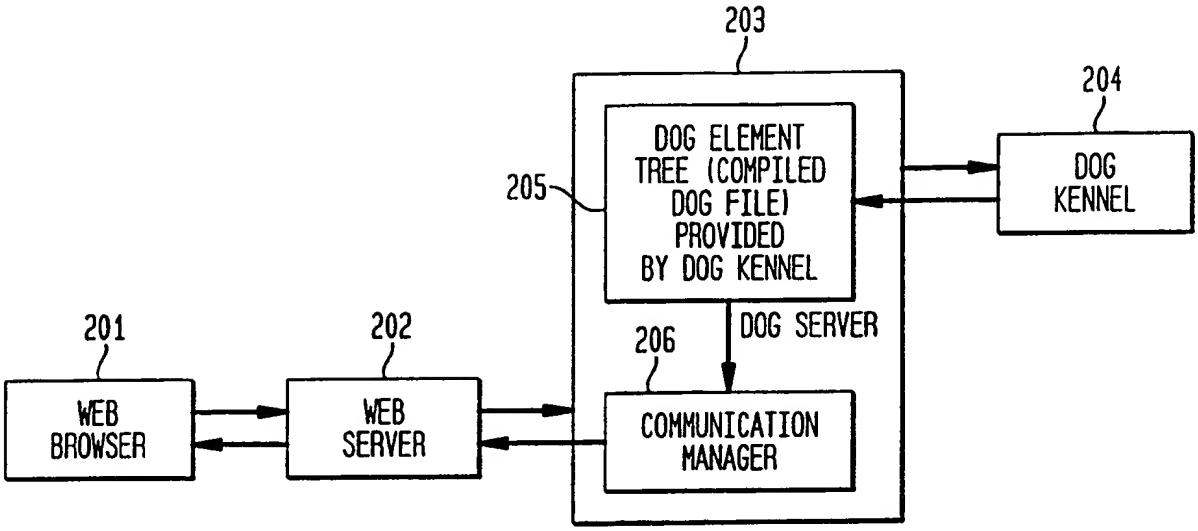

203
204
DOG ELEMENT TREE (COMPILED DOG FILE) PROVIDED BY DOG KENNEL
205
DOG KENNEL
201
202
206
DOG SERVER
WEB BROWSER
WEB SERVER
COMMUNICATION MANAGER

FIG. 3

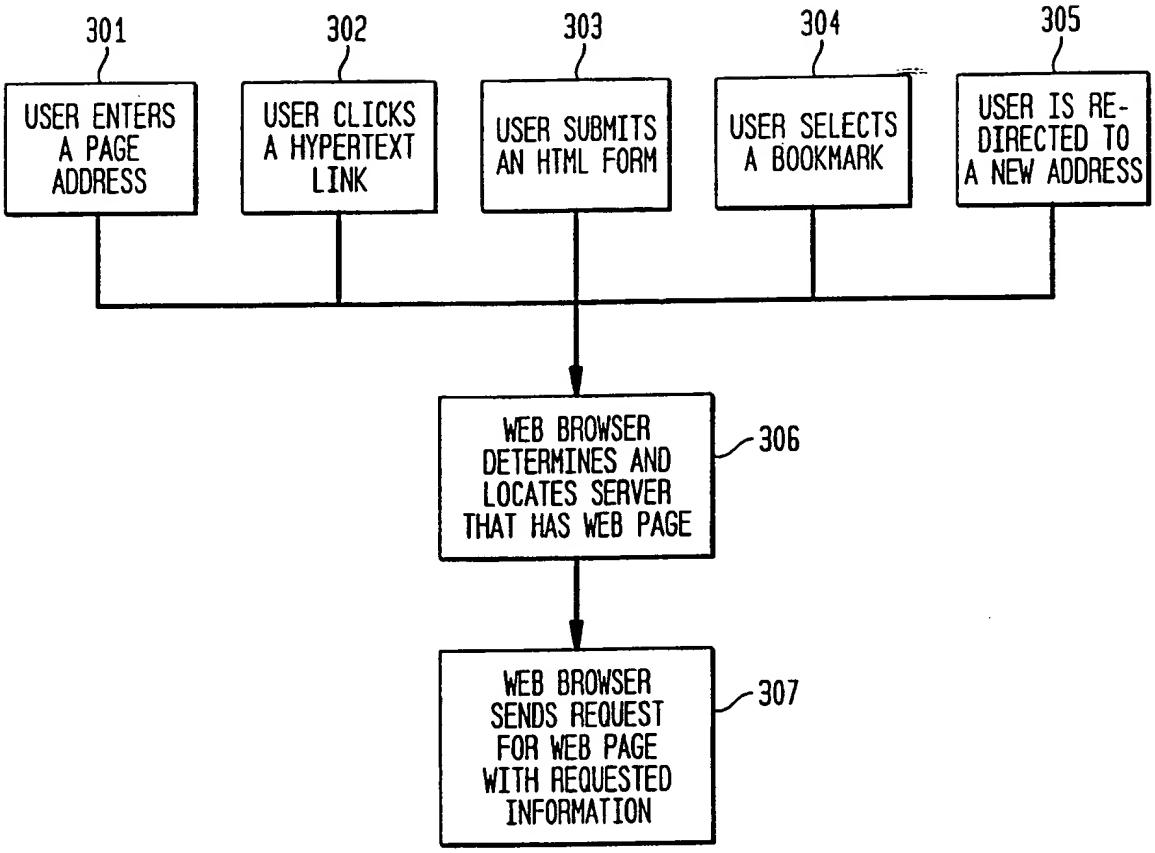

301
302
303
304
305
USER ENTERS A PAGE ADDRESS
USER CLICKS A HYPERTEXT LINK
USER SUBMITS AN HTML FORM
USER SELECTS A BOOKMARK
USER IS RE-DIRECTED TO A NEW ADDRESS
WEB BROWSER DETERMINES AND LOCATES SERVER THAT HAS WEB PAGE
306
WEB BROWSER SENDS REQUEST FOR WEB PAGE WITH REQUESTED INFORMATION
307

FIG. 4

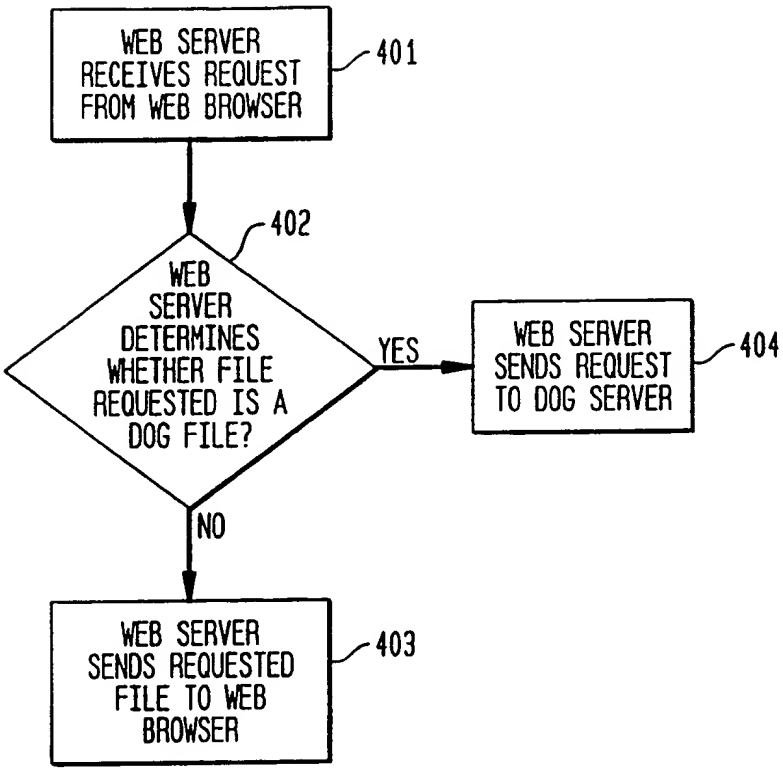

WEB SERVER RECEIVES REQUEST FROM WEB BROWSER
401
402
WEB SERVER DETERMINES WHETHER FILE REQUESTED IS A DOG FILE?
YES
WEB SERVER SENDS REQUEST TO DOG SERVER
404
NO
WEB SERVER SENDS REQUESTED FILE TO WEB BROWSER
403

FIG. 5A

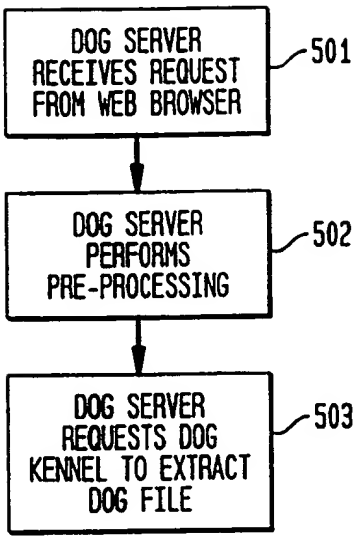

DOG SERVER RECEIVES REQUEST FROM WEB BROWSER
501
DOG SERVER PERFORMS PRE-PROCESSING
502
DOG SERVER REQUESTS DOG KENNEL TO EXTRACT DOG FILE
503

FIG. 5B

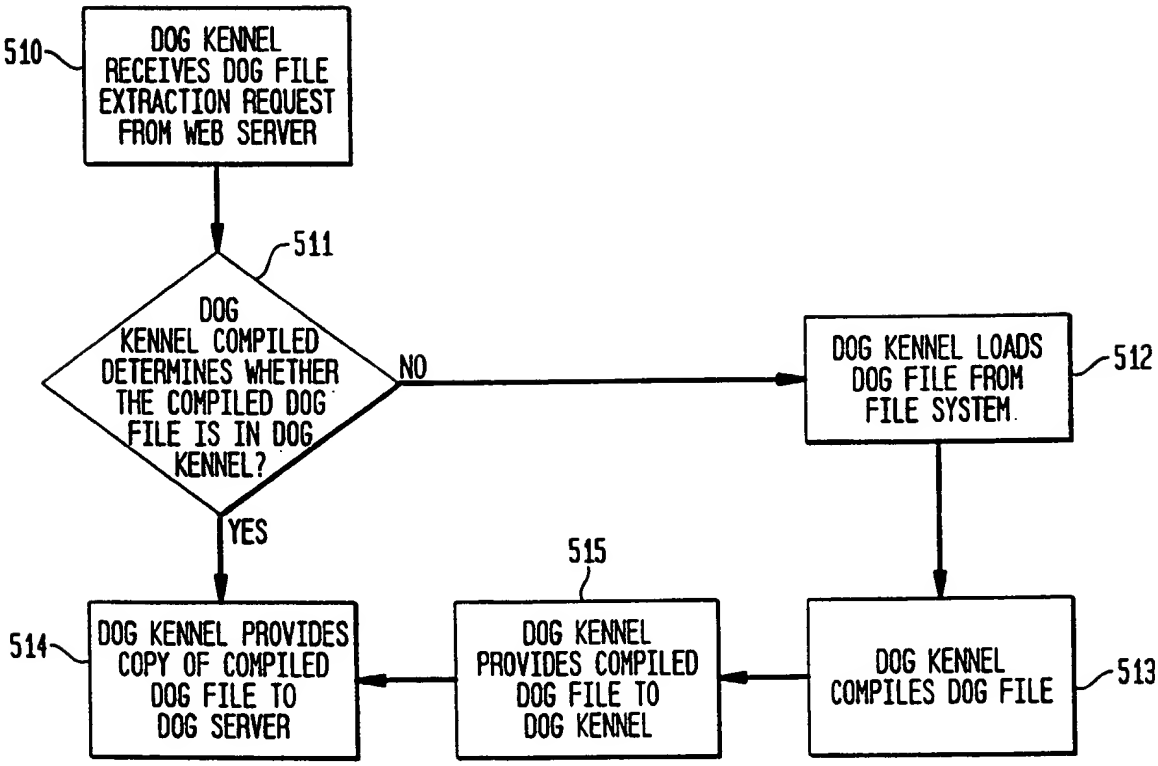

510
DOG KENNEL RECEIVES DOG FILE EXTRACTION REQUEST FROM WEB SERVER
511
DOG KENNEL COMPILED DETERMINES WHETHER THE COMPILED DOG FILE IS IN DOG KENNEL?
NO
YES
512
DOG KENNEL LOADS DOG FILE FROM FILE SYSTEM
513
DOG KENNEL COMPILES DOG FILE
515
DOG KENNEL PROVIDES COMPILED DOG FILE TO DOG KENNEL
514
DOG KENNEL PROVIDES COPY OF COMPILED DOG FILE TO DOG SERVER

FIG. 6

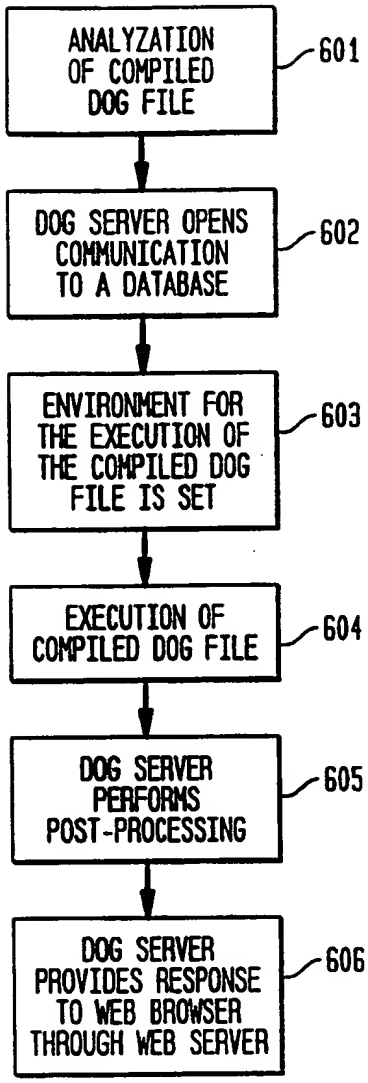

ANALYZATION OF COMPILED DOG FILE
601
DOG SERVER OPENS COMMUNICATION TO A DATABASE
602
ENVIRONMENT FOR THE EXECUTION OF THE COMPILED DOG FILE IS SET
603
EXECUTION OF COMPILED DOG FILE
604
DOG SERVER PERFORMS POST-PROCESSING
605
DOG SERVER PROVIDES RESPONSE TO WEB BROWSER THROUGH WEB SERVER
606

FIG. 7

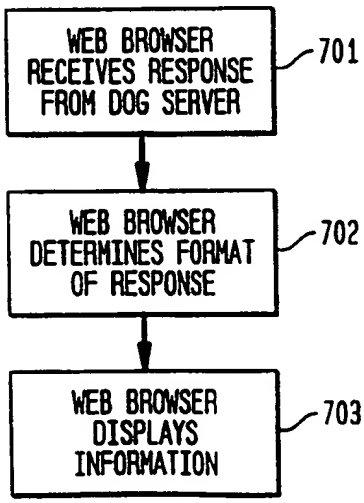
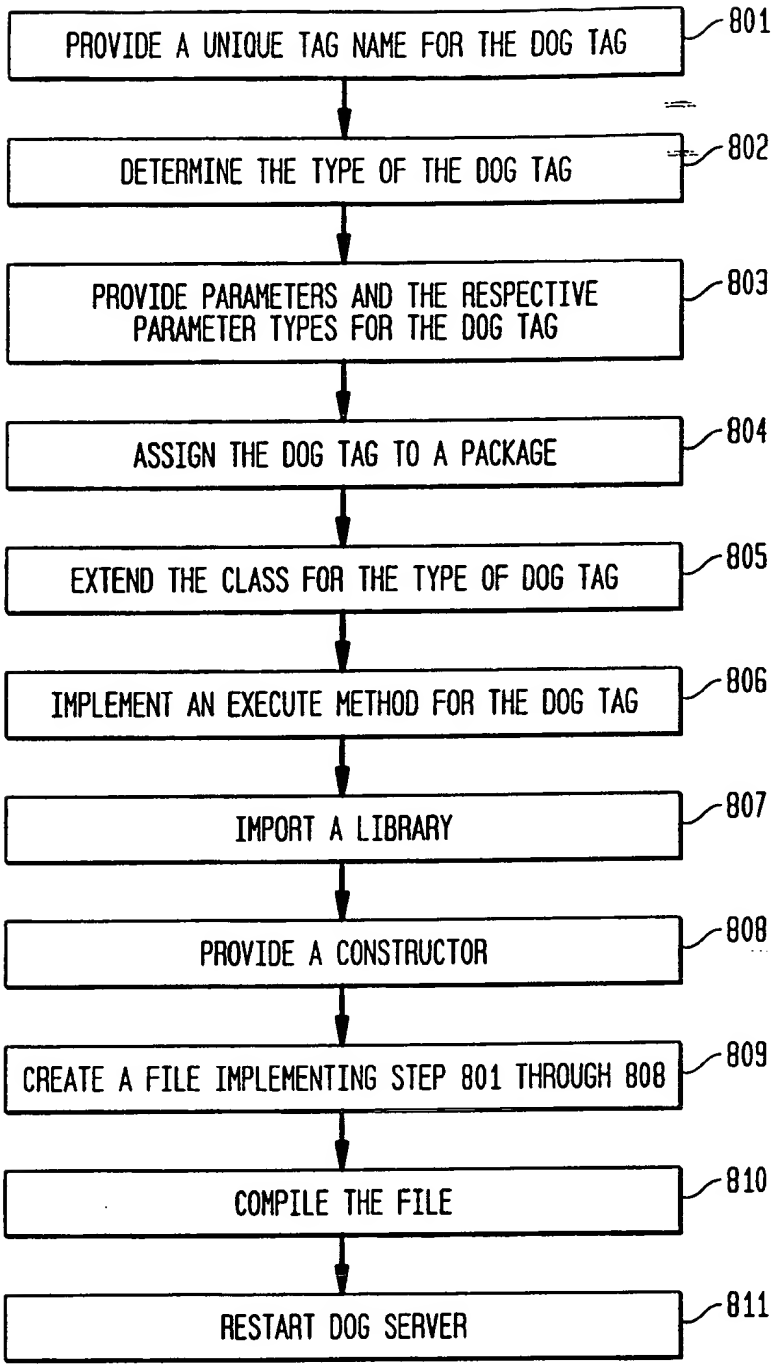

WEB BROWSER RECEIVES RESPONSE FROM DOG SERVER
701
WEB BROWSER DETERMINES FORMAT OF RESPONSE
702
WEB BROWSER DISPLAYS INFORMATION
703

FIG. 8


PROVIDE A UNIQUE TAG NAME FOR THE DOG TAG
801
DETERMINE THE TYPE OF THE DOG TAG
802
PROVIDE PARAMETERS AND THE RESPECTIVE PARAMETER TYPES FOR THE DOG TAG
803
ASSIGN THE DOG TAG TO A PACKAGE
804
EXTEND THE CLASS FOR THE TYPE OF DOG TAG
805
IMPLEMENT AN EXECUTE METHOD FOR THE DOG TAG
806
IMPORT A LIBRARY
807
PROVIDE A CONSTRUCTOR
808
CREATE A FILE IMPLEMENTING STEP 801 THROUGH 808
809
COMPILE THE FILE
810
RESTART DOG SERVER
811

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/25159

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 13/00
US CL :709/219; 707/104

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 709/219, 202, 203, 217, 302; 707/104, 103, 501

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EAST

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,623,656 A (LYONS) 22 April 1997, cols1-3.	1-20
Y	US 5,761,673 A (BOOKMAN ET AL) 02 June 1998, cols 1-5.	1-20
Y,P	US 5,894,554 A (LOWERY ET AL) 13 April 1999, cols 2-6.	1-20

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 04 JANUARY 2000	Date of mailing of the international search report 20 JAN 2000
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer VIET VU James R. Matthews Telephone No. (703) 305-9600

Form PCT/ISA/210 (second sheet)(July 1992)★

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☑ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.